ГЛАВА XX. Контроль качества и выбор модели

Правильная валидация существенно важнее правильной модели.

Owen Zhang

Если в вашей процедуре 10 параметров, вероятно, вы что-то упускаете.

А.Д. Перлис

После построения алгоритма машинного обучения возникает естественный вопрос – **как оценить качество / ошибку алгоритма?** Мы уже упоминали, что ошибка на обучении (train error) может сильно отличаться от ошибки на контрольной выборке (test error). Оценка ожидаемой ошибки важна для понимания пользы алгоритма, а также для выбора самого алгоритма и, даже более шире, выбора модели (Model Selection), в которой мы ищем алгоритм, и значения гиперпараметров. Ниже поговорим об этом подробнее. На рис. XX.1 показано качество алгоритма kNN при разных k – это иллюстрирует разницу между ошибками на обучении и контроле, а также необходимость правильного выбора значения гиперпараметра, в данном случае k (числа соседей).

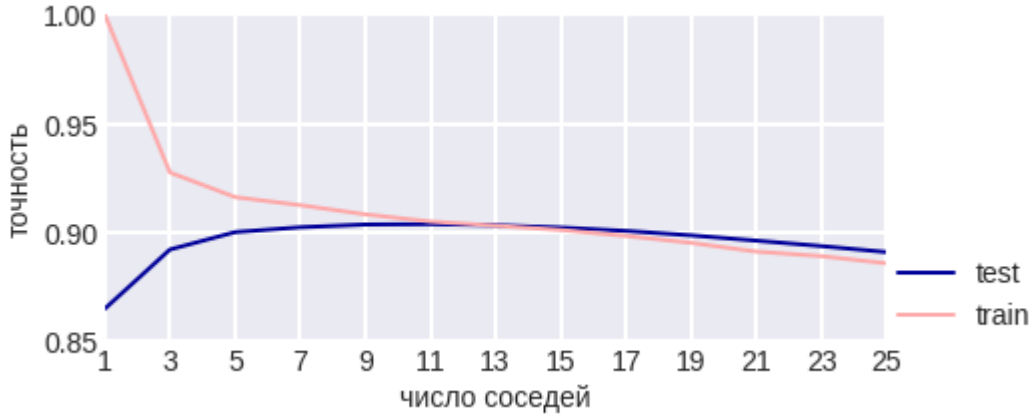


Рис. XX.1. Качество алгоритма на обучении и контроле при разных значениях гиперпараметра.

Заметим, что максимальное качество на обучающей выборке при $k=1$, однако для рассмотренных значений этого гиперпараметра это соответствует минимальному качеству на контрольной выборке.

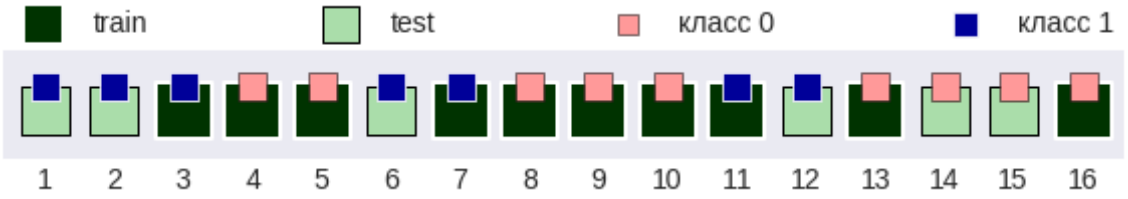
Отложенный контроль (hold-out / held-out / validation data)

Базовая идея контроля качества очень проста: данные нужно разделить на две части:

- **обучающая часть** (обучающая выборка или, по-простому, «обучение») – для обучения алгоритма,
- **тестовая часть (контрольная выборка, отложенный контроль)** – для оценки качества обученного алгоритма. Этот процесс будем называть **тестированием** или **контролем качества**.

На рис. XX.2 показаны подобные разбиения[1]: случайное (также говорят «с перемешиванием» объектов) и неслучайное (первая часть выборки – обучение, вторая – тест). Здесь и далее тёмно-зелёный цвет соответствует обучающей выборке, светло-зелёный – контрольной. Также может быть дана некоторая метаинформация, на рис. XX.2 – указаны метки объектов (синим цветом – класс 1, розовым – класс 0).

```
X_train, X_test, y_train, y_test =
    sklearn.model_selection.train_test_split(x, y,
                                             test_size=0.33,
                                             random_state=41)
```




```
X_train, X_test, y_train, y_test =
    sklearn.model_selection.train_test_split(x, y,
                                             test_size=0.33,
                                             shuffle=False)
```

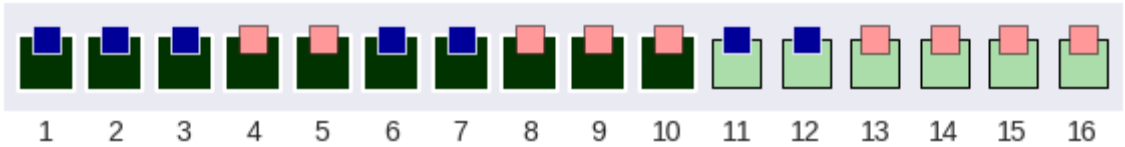


Рис. XX.2. Пример отложенного контроля.

[1] Обычно мы избегаем указания фрагментов кода в основном изложении, т.к. это сильно привязывает внимание читателя к конкретному языку программирования и используемой библиотеке. В этой главе всё-таки отходим от этого правила. Кстати, Функция **train_test_split**, не смотря на конкретное название, хорошо подходит для разбиения данных на две части (не обязательно для обучения и тестирования).

Недостатки отложенного контроля:

- оценка ошибки зависит от конкретной выбранной контрольной (отложенной) выборки, часто сильно меняется при другом выборе,
- если переобучить[1] алгоритм на всех данных, то мы не знаем оценку его ошибки (это уже формально другой алгоритм). Переобучивать разумно, т.к. мы хотим использовать все данные для построения итогового алгоритма (это, как правило, повышает его качество).

Два алгоритма могут называться одинаково, например, kNN. Но при разных обучающих выборках они разные!

Последний недостаток, в некотором смысле, неустранимый: чтобы оценить качество алгоритма надо использовать часть выборки, а если алгоритм обучен на всех имеющихся данных, то оценить качество с помощью сравнения его ответов с истинными невозможно (нужны новые размеченные данные). Обычно на отложенный контроль выбирают около 20% от всей выборки. Чем больше отложенный контроль, тем объективнее оценка качества алгоритма, но обучающая выборка становится меньше (и алгоритм может ухудшаться). Если алгоритм потом переучивать на всей выборке, то чем меньше отложенный контроль, тем сильнее он [алгоритм] будет похож на тот, который мы оценивали.

Теперь, когда мы познакомились с базовым способом контроля качества, стоит сформулировать **три правила разбиения выборки**:

- тест / валидация[2] моделирует реальную работу алгоритма,
- нельзя явно или неявно использовать метки объектов, на которых оцениваешь ошибку[3] (качество),
- тест / валидация должны быть случайными (или специально подготовленным).

Поясним первое правило. Например, пусть описания объектов в задаче – информация о пользователях в какие-то моменты времени, при этом об одном и том же пользователе может быть несколько записей в разные моменты времени. Пусть также наш алгоритм должен работать на новых пользователях

[1] Здесь «переобучить» – взять и обучить на новых данных.

[2] Что такое «валидация», скажем позже.

[3] Это называется «утечкой меток».

(информации о которых не было в обучении), тогда и в отложенной выборке должны быть такие новые пользователи (информации о которых не было в обучении). Это нужно учитывать при разбиении выборки.

Ещё иллюстрация первого правила: если на практике со временем пропорции классов сохраняются, то и при тесте они должны сохраняться (в тесте такие же, как в обучении). Часто распределения в обучении и тесте должны быть одинаковыми (по всем признакам, в том числе по целевому признаку).

Первое правило появилось из-за некоторых довольно частых проблем данных: вхождения целых однородных групп в обучающую выборку (например, записей о конкретном пользователе), наличия дубликатов (абсолютно одинаковых объектов) или «почти дубликатов». В исходной выборке из-за ошибок дубликаты есть, при простом случайном делении есть опасность, что одинаковые по описаниям объекты попадут одновременно в обучающую и отложенную выборки, это сделает оценку качества оптимистичной (завышенной), поэтому такого нельзя допускать.

Всегда думайте, как алгоритм будет использоваться.

Второе правило проиллюстрируем классическим примером из курса Тибширани[1]: нельзя найти подмножество признаков, максимально коррелирующих с целевым, а потом оценить ошибку алгоритма, например с помощью отложенного контроля на этом подмножестве признаков. Такая оценка будет неверной, мы несколько раз использовали метки наших объектов: при выборе признаков и при обучении!

Нельзя смотреть на метки несколько раз!

Для иллюстрации разумности третьего правила вспомним рис. XX.2. Объекты могут располагаться в исходной выборке не случайно, а согласно некоторому порядку. Например, сначала идут объекты класса 0, а потом – класса 1 в бинарной задаче классификации. В этом случае может получиться, что при разбиении выборки на обучение и тест, одна из подвыборок будет целиком состоять из объектов одного из классов[2]. Если же объекты расположены по времени появления, то возможно их и не надо перемешивать. Первые «старые» объекты попадут в обучение,

По умолчанию всегда применяют случайное разбиение на обучение и тест.

[1] James G. et al. An introduction to statistical learning. – New York : Springer, 2013.

[2] Для следования третьему правилу в библиотеке scikit-learn в большинстве функций для разбиения данных есть соответствующий параметр перемешивания выборки: **shuffle=True**.

а вторые «новые» – в тест (если мы делаем алгоритм прогнозирования и это соотносится с первым правилом).

Кроме отложенного контроля, основанного на базовой идеи разделения выборки, есть и другие способы контроля качества алгоритма, которые мы дальше продемонстрируем, пока перечислим основные:

- отложенный контроль (held-out data, hold-out set),
- скользящий контроль / перекрёстная проверка[1] (cross-validation),
- бутстреп (bootstrap),
- контроль по времени (out-of-time-контроль).

Варианты отложенного контроля

При многократном применении разбиения **Random Subsampling Cross-Validation (ShuffleSplit)** k раз случайно выбирается отложенный контроль, а затем усредняются ошибки на всех отложенных выборках, см. рис. XX.3.

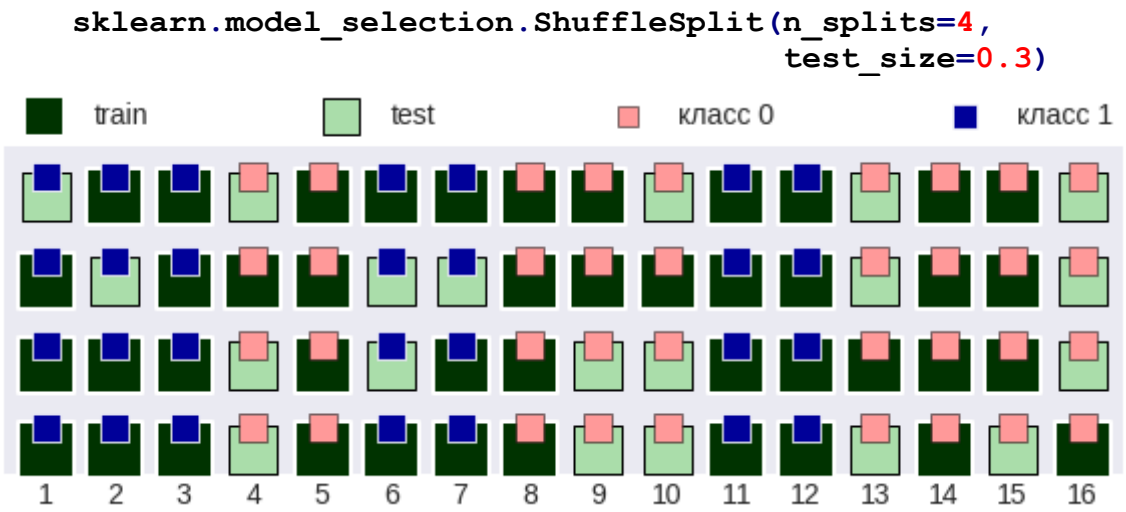


Рис. XX.3. Пример многократного отложенного контроля.

Есть аналогичный способ **контроля без разбиения групп Shuffle-Group(s)-Out Split** – когда вводится специальный служебный категориальный признак «группа» и, таким образом, все объекты распадаются на разные группы, а при

[1] Также термином «скользящий контроль» называют все указанные способы контроля. Но вот цитата с Wiki: «many sources instead classify holdout as a type of simple validation, rather than a simple or degenerate form of cross-validation ».

разбиении на обучение и тест каждая группа попадает целиком или в обучение или в тест, см. рис. XX.4. Выше был пример, поясняющий для чего это нужно.

```
sklearn.model_selection.GroupShuffleSplit(n_splits=4,
                                          test_size=0.3)
for itrain, itest in cv.split(x, groups=g):
```

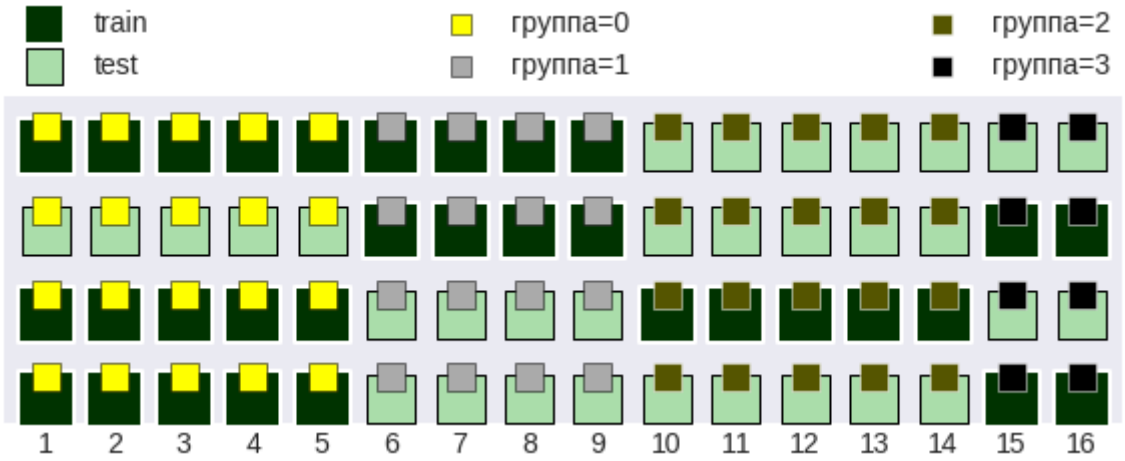


Рис. XX.4. Пример многократного отложенного контроля без разбиения групп.

```
sklearn.model_selection.StratifiedShuffleSplit(n_splits=4,
                                               test_size=0.3)
for itrain, itest in cv.split(x, groups=y):
```

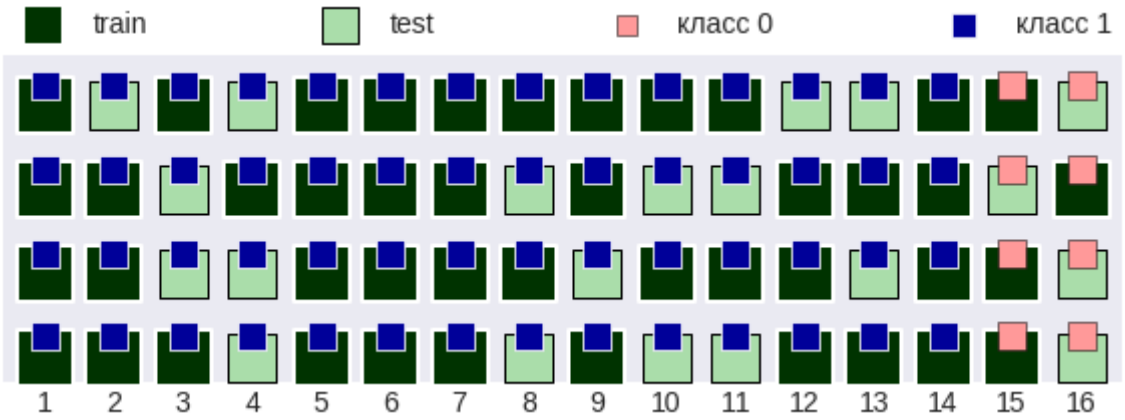


Рис. XX.5. Пример многократного отложенного контроля без разбиения групп.

Есть также способ **контроля со случайными разбиениями с сохранением пропорции классов (стратифицированного контроля)**, см. рис. XX.5 – здесь всего два объекта класса 0, но при разбиениях на обучение и тест один гарантированно попадает в обучение, а второй – в тест. В общем случае разбиение старается сохранить пропорции классов в обучении и тесте. Как видим, разные способы разбиения (и соответствующие методы в современных библиотеках) нужны для выполнения описанных выше правил разбиения.

Бутстреп (Bootstrap)

Это контроль с помощью выбора с возвращением из всей выборки m объектов. Так формируется подвыборка полного объёма m, на которой производится обучение модели[1], на остальных объектах (которые не попали в обучение) – контроль. На рис. XX.6 показан набор индексов **`i_train`** объектов, отобранных в обучение, и набор остальных индексов **`i_test`** объектов, отобранных для валидации.

Заметим, что в обучении объекты могут дублироваться (т.е. попадает несколько копий одного и того же объекта). Это нужно учитывать при использовании бутстрепа (часто это позволяет даже строить алгоритмы более высокого качества, например в случайных лесах, см. главу про случайные леса). В контроль попадает примерно

$$\left(1-\frac{1}{m}\right)^m \approx e^{-1} \approx 0.37 = 37\% \text{ выборки}^2.$$

```
i_train = [9, 16, 14, 9, 7, 12, 3, 12, 9, 8, 3, 2, 16, 12, 6, 16]
i_test =  [1, 4, 5, 10, 11, 13, 15]
```

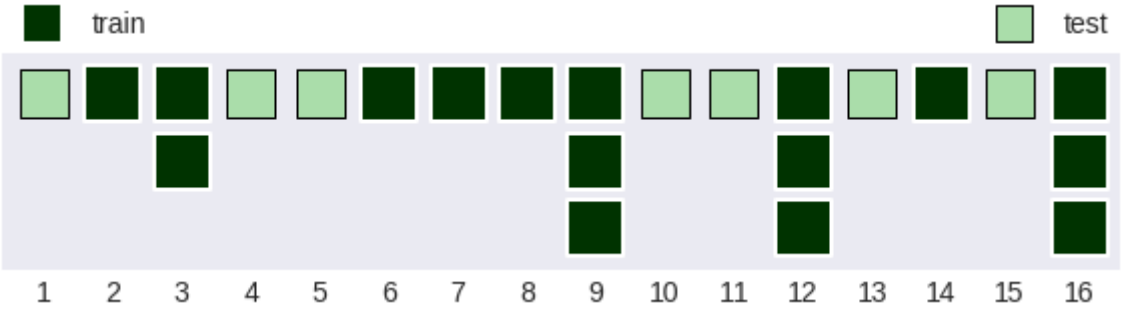


Рис. XX.6. Пример бутстрепа.

Важное положительное свойство бутстрепа – модель учится на выборке того же объёма, что и итоговая (которую мы обучим по всей), но при этом использует не все данные (за счёт дубликатов)! С точки зрения распределения бутстреп-выборка похожа на исходную.

[1] Можно представить, что у нас есть корзина с m шариками, которые пронумерованы от 1 до m. Мы m раз перемешиваем шарики, вынимаем случайный шар, запоминаем его номер (объект с таким номером откладывается в обучение) и возвращаем шар в корзину.

[2] Эта формула, по сути, вероятность того, что конкретный шар ни разу не будет извлечён из корзины. Вероятность извлечения – $1/m$, не извлечения – $(1 - 1/m)$.

Перекрёстная проверка по фолдам (*k*-fold cross-validation)

Опишем схему контроля по фолдам, которую в литературе часто сокращают как «k-fold CV» или «k-CV»:

- разделить выборку на k примерно[1] равных частей (они и называются «фолдами»),
- цикл по $i = 1 \ldots k$
    - использовать i-ю часть для теста (вычислить ошибку на ней), а объединение остальных – для обучения,
- усреднить k ошибок, вычисленных на разных итерациях цикла (можно использовать их дисперсию для оценки доверия к полученному качеству).

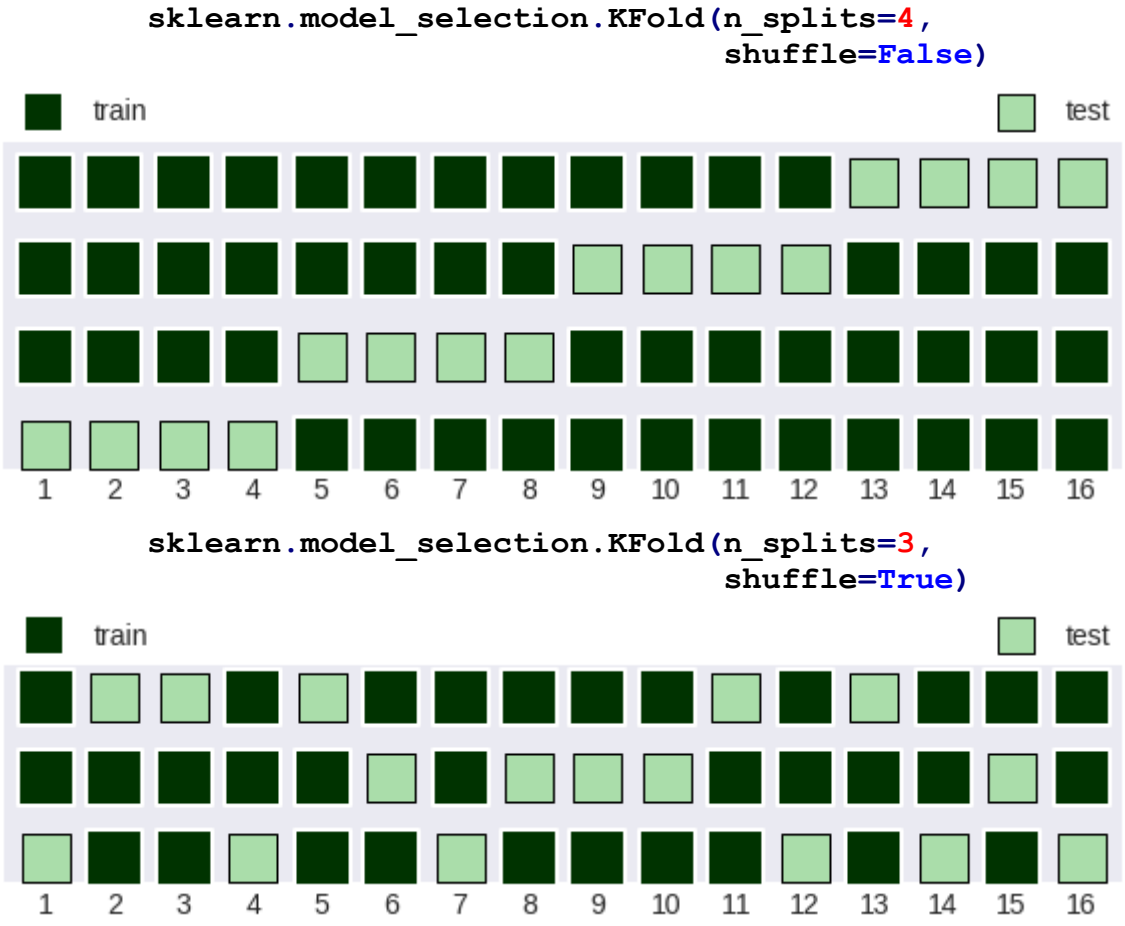


Рис. XX.7. Перекрёстная проверка по фолдам (k-fold cross-validation).

На рис. XX.7 показаны примеры k-fold CV: когда делим на 4 части последовательно (здесь они определяются порядком объектов) и на 3 части с

[1] Так как не всегда возможно деление на равные части. На рис. XX.7 сначала 16 = 4 + 4 + 4 + 4 и деление происходит на равные части, затем 16 = 5 + 5 + 6 и последний фолд получается больше.

предварительным случайным перемешиванием объектов[1] (тогда фолды случайные). Обратите внимание, что каждый объект ровно один раз попадает в тест и $(k-1)$ раз – в обучение.

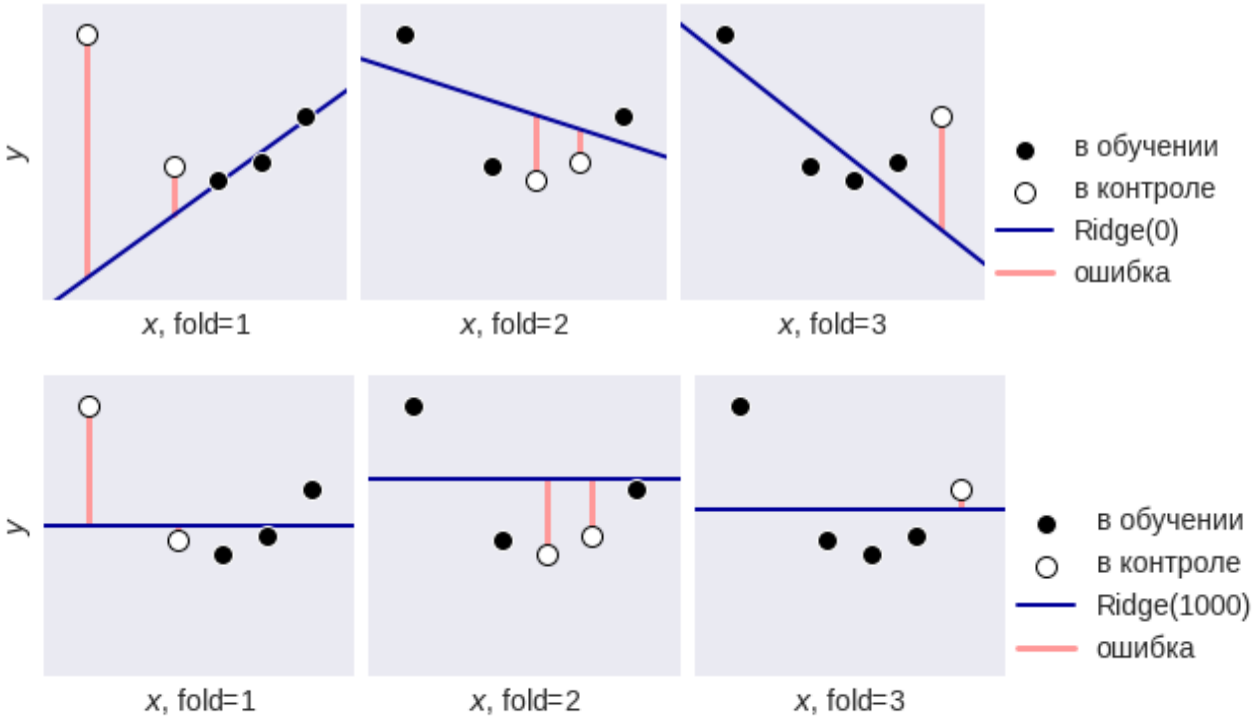


Рис. XX.8. Перекрёстная проверка по фолдам: сравнение линейной и константной регрессий.

На рис. XX.8 показан модельный пример: в выборке $m=5$ точки, число фолдов $k=3$, сравниваем гребневую регрессию без регуляризации Ridge(0)[2] (верхний ряд) и с большой регуляризацией Ridge(1000) (нижний ряд). Первая регрессия строит прямую по обучению (чёрные точки), а вторая является почти константным решением, которое соответствует усреднению целевых значений точек обучения (их y-координат). Визуально константный алгоритм лучше линейной регрессии, т.к. средняя ошибка (средняя длина розовых отрезков) ниже. Видим, что здесь фолды организованы по увеличению x-координаты точек обучения, что нарушает одно из золотых правил организации контроля.

Обычно в перекрёстной проверке по фолдам стараются выбирать число фолдов $k=10$. При больших значениях k надёжнее оценка качества (обучающая выборка больше походит на все данные), но и время контроля возрастает. Также отметим, что при больших k контроль может быть довольно малым по

[1] Параметр **shuffle=True** отвечает за перемешивание выборки.

[2] Это соответствует инициализации соответствующего объекта в библиотеке scikit-learn: число в скобках – параметр регуляризации.

объёму, а не все ошибки/качество адекватно оцениваются на малых выборках. Например, в малых случайных выборках высока вероятность совпадения меток всех объектов в задаче классификации, но с этим позволяет справится стратифицированная версия k-CV – с сохранением пропорции классов, см. рис. XX.9.

```
sklearn.model_selection.StratifiedKFold(n_splits=4,
                                        shuffle=False)
```

Рис. XX.9. Стратифицированная перекрёстная проверка по фолдам.

На рис. XX.10 показана версия перекрёстной проверки по фолдам без разбиения групп. Есть также возможность делать заданные разбиения[1]: когда мы сами определяем, как именно делать k разбиений, указывая служебный категориальный признак с k категориями.

```
sklearn.model_selection.GroupKFold(n_splits=3)
```

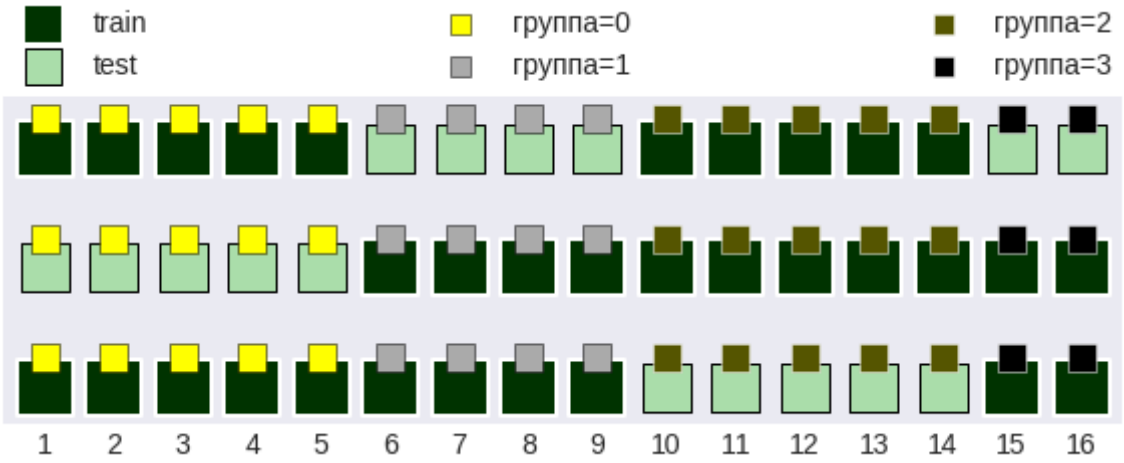


Рис. XX.10. Перекрёстная проверка по фолдам без разбиения групп.

[1] В sklearn функция sklearn.model_selection.PredefinedSplit.

Проверка «по одному» (Leave-one out cross-validation, LOOCV)

Проверка «по одному» – это перекрёстная проверка по фолдам при $k = m$ (когда число фолдов совпадает с числом объектов). Здесь проводится $k = m$ разбивок, в каждой тестовая выборка состоит лишь из одного объекта. На рис. XX.8 изображён как раз такой способ контроля. Ясно, что это вырожденный случай, который может долго вычисляться, поэтому на практике почти не используется, но часто применяется в теоретических работах, поскольку часто оценить ошибку на одном объекте довольно легко (иногда можно даже аналитически).

`sklearn.model_selection.LeaveOneOut`

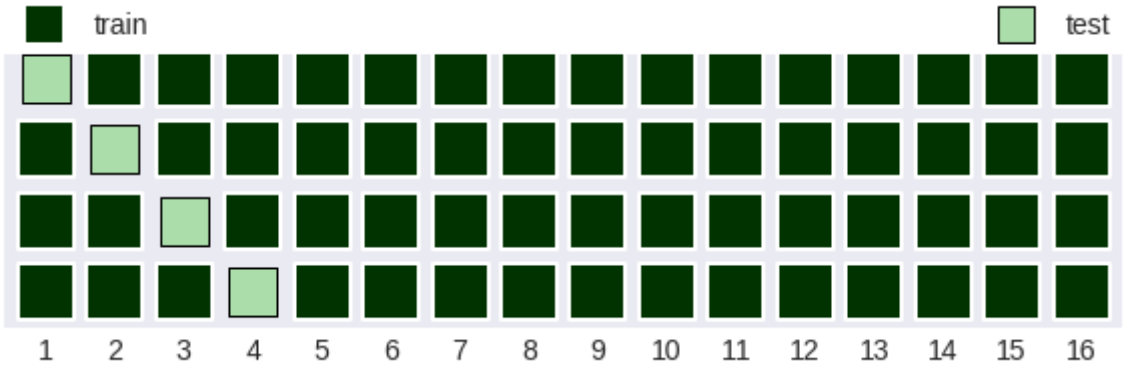


Рис. XX.11. Проверка «по одному» (показаны первые 4 проверки).

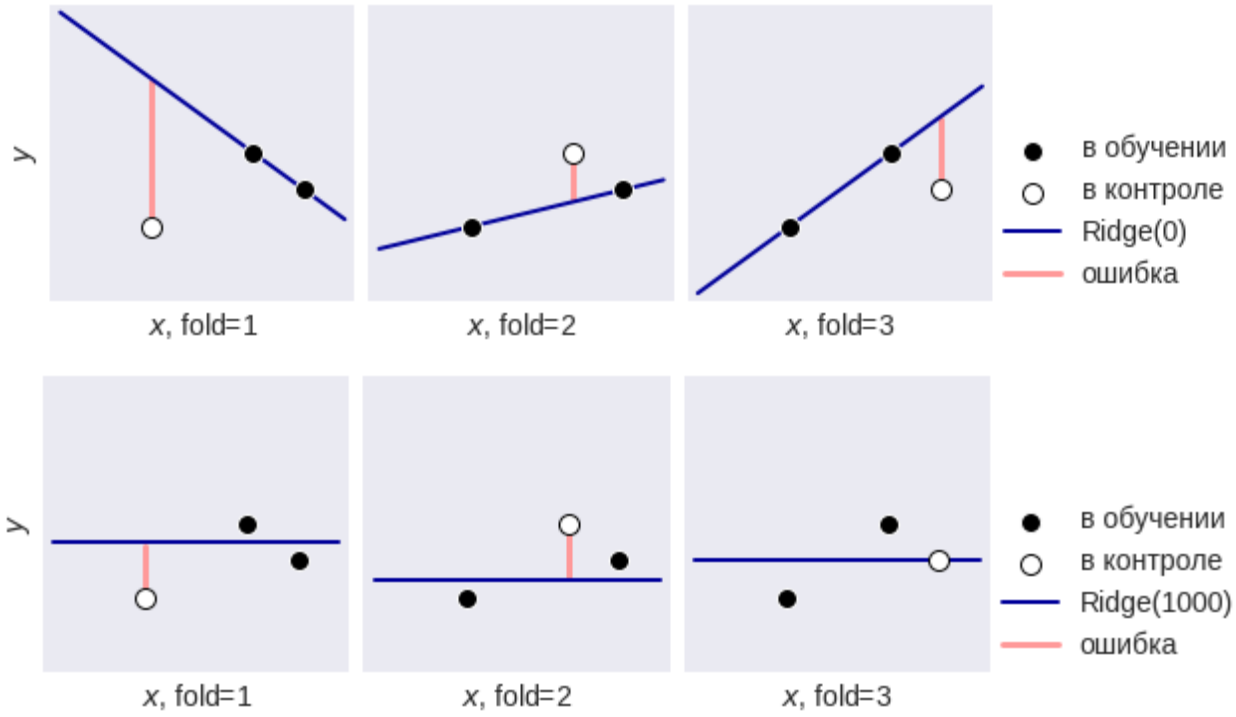


Рис. XX.12. Проверка по одному:
сравнение линейной и константной регрессий.

На рис. XX.11 показаны только первые 4 разбивки[1] при LOO, а на рис. XX.12 сравнение регрессий методом LOO, аналогичное тому, что мы показывали на рис. XX.8. Также полезно в этом контексте вспомнить аналогичный метод в статистике: «Складной нож» (jackknife), который используют для оценки параметров следующим образом:

$$\overline{x}_{-i} = \frac{1}{m-1}\sum_{j\neq i} x_j \quad \text{(оценка среднего на выборке без одного объекта)},$$

$$\overline{x} = \frac{1}{m}\sum_{i=1}^{m} \overline{x}_{-i} \quad \text{(итоговая оценка среднего)},$$

$$\overline{\text{var}} = \frac{m-1}{m}\sum_{i=1}^{m} (\overline{x} - \overline{x}_{-i})^2 \quad \text{(оценка дисперсии оценки)}.$$

Хотя на практике чаще используют бутстреп-оценки.

Контроль по группам (LeaveOneGroupOut)

В контроле по группам задаётся служебный категориальный признак «группы», содержащий k категорий. Производится k отложенных контролей, в каждом отложенная выборка состоит из объектов соответствующей группы, см. рис. XX.13.

```
from sklearn.model_selection import LeaveOneGroupOut
```

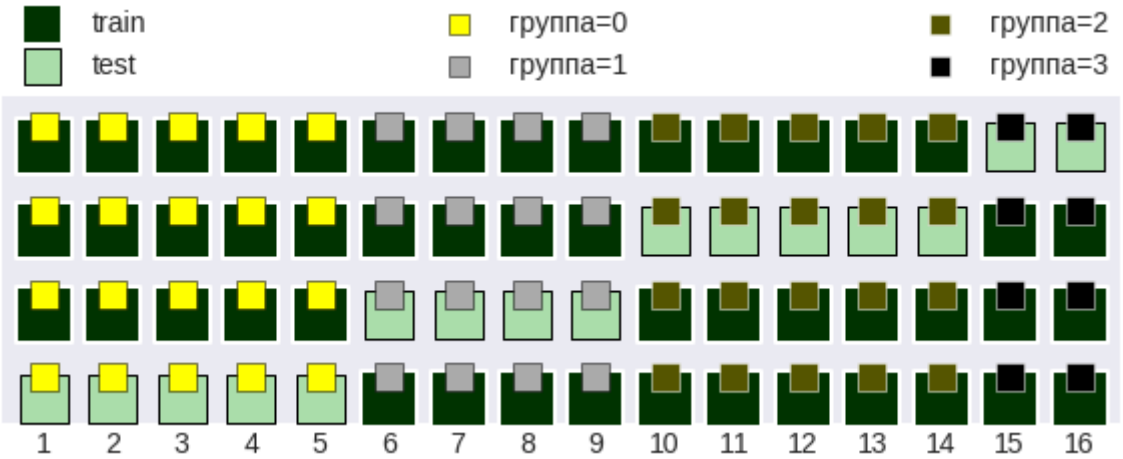


Рис. XX.13. Проверка «по группам».

[1] В sklearn ещё есть sklearn.model_selection.LeavePOut – всевозможные n-ки используются для контроля.

Ошибку алгоритма можно оценить как среднее ошибок на всех группах, но чаще ошибку оценивают с учётом мощностей групп:

$$e = \sum_{t=1}^{k} \frac{m_t}{m} e_t ,$$

где m_t – мощность t-й группы.

Контроль по времени (Out-of-time-контроль)

В случае, если данные представляют временной ряд, используют специальный метод разбиения временных рядов – **TimeSeriesSplit (Time series cross-validation**). Он показан на рис. XX.14 – мы пытаемся обучиться на данных из прошлого и предсказать целевое значение на группе данных из будущего. Часто из-за ограниченного объёма выборки при тесте по времени не удаётся провести много отложенных контролей. Здесь также не забываем о правилах разбиения выборки для контроля, например, если разрабатывается алгоритм прогнозирования продаж на неделе, результатами которого пользуются по утрам в понедельники, то «отщеплять» для контроля нужно по целым календарным неделям.

```
from sklearn.model_selection import TimeSeriesSplit
TimeSeriesSplit (n_splits=4, # сколько делить
                 max_train_size=None, # сколько делить
                 test_size=None, # n_samples // (n_splits + 1)
                                 # если gap=0
                 gap=0) # сколько "откусывать" в конце
                        # по умолчанию не используется
```

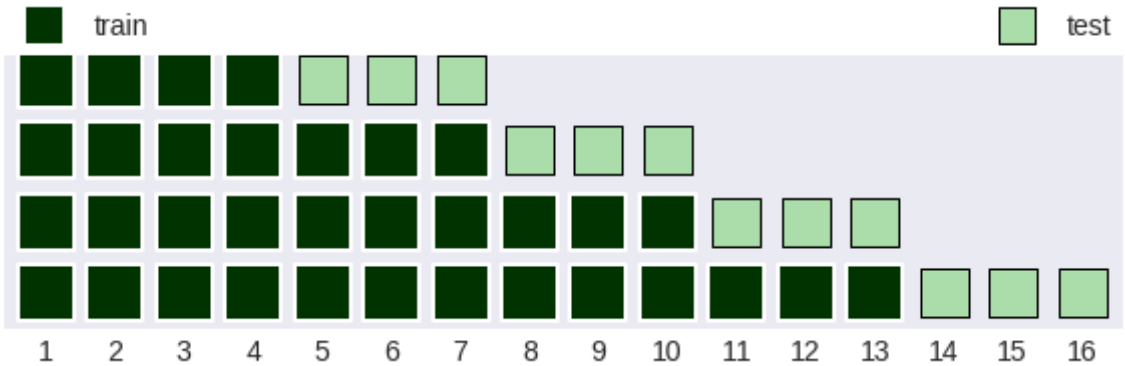


Рис. XX.14. Контроль по времени.

Часто контроль следует организовывать так, чтобы прогнозировать события не с текущего момента времени, а через некоторый интервал времени (параметр gap), это может быть связано, например, с тем, что данные приходят с

запаздыванием. На рис. XX.15 показана организация контроля прогноза трёх меток с зазором $\text{gap}=2$ и ограничением на объём обучения 7 (например, можно использовать только данные последней недели), т.е. по объектам с временными номерами $(\max[t-2-7,1],\ldots,t-2-1)$ предсказываются метки с номерами $(t,t+1,t+3)$.

```
TimeSeriesSplit(n_splits=3, test_size=3, max_train_size=7, gap=2)
```

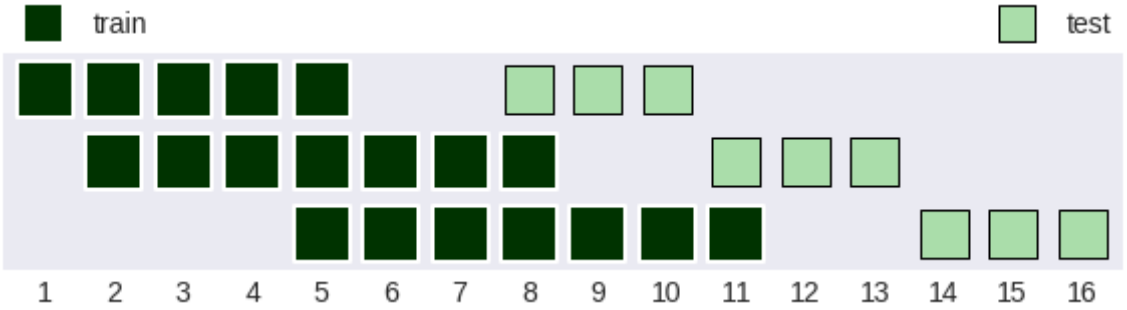


Рис. XX.15. Контроль по времени с зазором и ограничением обучения.

Мы уже познакомились в примерах с ситуациями, когда контроль организовывается так, что информация о конкретном пользователе попадает целиком в обучение или отложенный контроль. Такая форма контроля называется **«out of sample»**. Форму контроля на данных «из будущего» относительно времени обучающих данных называют **«out of time»**. В таблице ниже показаны разные варианты проверки качества алгоритма. Например «out of sample, out of time» означает проверку на новых пользователях в будущем.

user index		
	Training Data	in sample out of time
	out of sample in time	out of sample out of time

time

Тестирование и валидация

Проблема контроля качества, о которой мы до настоящего момента говорили, не единственная, которую приходится решить исследователю. Чтобы оценить качество алгоритма, его надо предварительно выбрать, поэтому возникает проблема **выбора модели (Model Selection)**[1] и настройки (выбора

[1] Заметим, что приводимый код использовал модуль model_selection.

значений) гиперпараметров алгоритма. Проблема выбора модели заключается в том, какое параметрическое семейство алгоритмов использовать для решения конкретной задачи. Моделей потенциально очень много (но конечное число): kNN, SVM, Linear, NN и т.п. Алгоритм имеет (обычные) **параметры (model parameters)** – они настраиваются в процессе обучения[1] (т.е. в результате решения оптимизационной задачи с учётом обучающих данных) и **гиперпараметры (hyperparameters)** – их значения выбираются до обучения, экспертно или переборно, как раз с помощью процедур, о которых пойдёт речь в этой главе. Например, в полиномиальной регрессии степень полинома это гиперпараметр (должна быть задана до обучения), а коэффициенты полинома – параметры (определяются при обучении).

Можно, допустив некоторую вольность, сказать, что с помощью гиперпараметра модель разбивается на разные подмодели. Например, kNN при разном выборе k: 1NN, 3NN, 5NN.

Между прочим, проблема выбора модели есть не только для размеченных данных, но и для неразмеченных (USL).

В широком смысле проблема выбора модели (Model Selection) заключается не только в выборе модели алгоритма, а всего «пайплайна решающего задачу»:

- выбор модели алгоритмов,
- выбор значения гиперпараметров,
- выбор признаков,
- выбор способа предобработки данных (заполнения пропусков, детектирования и удаления выбросов и т.п.).

Покажем, как базовая идея контроля качества трансформируется для решения задач выбора модели и оценке её качества. Все имеющиеся данные (в задаче обучения с учителем – все размеченные данные) делят на три части, см. рис. XX.16:

- **Обучающая выборка (Training Set)** – на этом множестве производится обучение модели (т.е. настройка её параметров).
- **Валидационная выборка (Validation Set)** – здесь как раз производится выбор пайплайна (выбор модели, гиперпараметров модели, признакового пространства и т.п.), иногда эту выборку и процесс определения качества на ней называют **локальным контролем**.

[1] При использовании библиотеки sklearn это происходит внутри метода fit.

- **Тестовая выборка (Test Set)** – здесь производится итоговая оценка качества выбранного пайплайна.

Рис. XX.16. Схематическое разбиение исходной выборки на обучающую, валидационную и тестовую.

При валидации мы разные пайплайны (алгоритмы) обучаем на обучающей выборке и вычисляем их качество на валидационной выборке, выбирается пайплайн (алгоритм) с максимальным качеством (если это единственное требование, также можно учитывать другие требования к алгоритму: время работы, компактность и т.п.), затем он тестируется на отдельной тестовой выборке. Доверять качеству на валидации не стоит, т.к. мы выбирали максимальное[1]. Например, если показатели качества разных алгоритмов случайные с одинаковым среднем, то выбор максимального даёт оптимистическую оценку итогового качества.

Иногда (ограниченный набор данных, небольшое число пайплайнов, отобранных экспертами, большая валидационная выборка и т.п.) доверяют оценке качества на валидации.

Аналогично, можно модифицировать и остальные способы контроля. Например, валидацию (выбор модели) делать с помощью k-Fold CV, а тестирование с помощью отложенной выборки, т.е. разбиваем данные на две части, на первой делаем k-Fold CV для разных алгоритмов и выбираем лучший, на второй оцениваем его качество.

На рис. XX.17 показаны графики зависимости качества логистической регрессии от параметра регуляризации в двух схожих задачах. Как видно, кривые для k-Fold CV при разном числе фолдов похожи, по крайней мере, максимумы достигаются в близких точках. В первой задаче эти кривые похожи на кривую качества на тесте, во второй – нет. Поэтому **не всегда хорошая валидация гарантирует наилучшую работу алгоритма на конкретном тесте.**

[1] Можно провести аналогию с подбрасыванием монеты, в которой вероятность выпадения орла соответствует вероятности верного ответа алгоритма. У нас есть набор нечестных монеток, мы каждую бросили 10 раз. В результате получили такие вероятности: 0.4, 0.5, 0.55, 0.6, 0.65. Можно выбрать монету с максимальной вероятностью, но лучше её ещё побросать... Заметим, что если бы бросков было не 10, а 1000, то мы могли бы довериться вероятности 0.65.

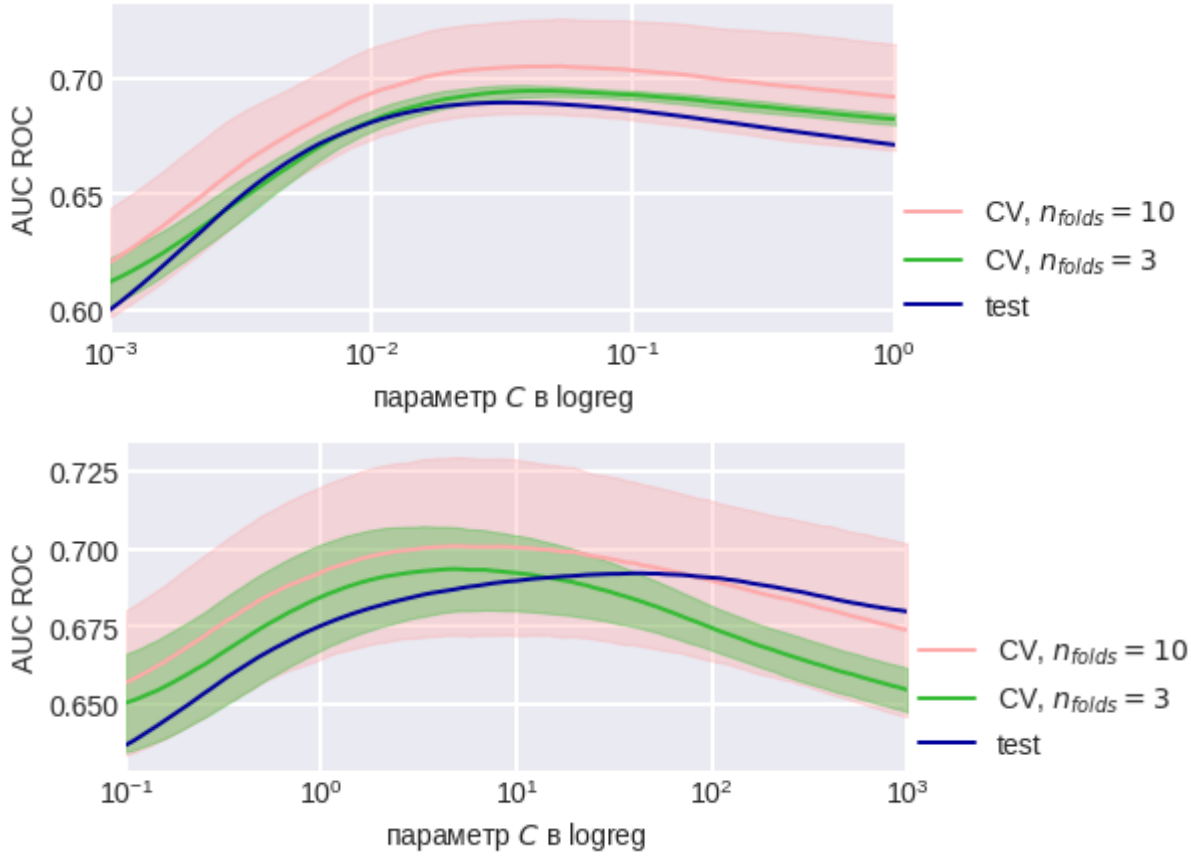


Рис. XX.17. Две похожие задачи: есть и нет согласование CV / test.

Локальный контроль

Локальный контроль – это организация валидации в случае, когда итоговое качество алгоритма будет оцениваться на заранее заданной выборке (глобальном контроле). Локальный контроль используется, в первую очередь, при участии в соревнованиях по машинному обучению и при обучении с частичной разметкой (semi-supervised learning). Например, в соревнованиях глобальный контроль часто заранее известен. При локальном контроле вспоминаем наши правила разбиения выборок. Распределения по признакам на локальном и глобальном контролях должны совпадать, в том числе, распределение выбросов, пропусков и т.п. При прогнозировании в локальном контроле мы должны прогнозировать на схожий с глобальным период (с того же дня недели, столько же праздников впереди и т.п.). На рис. XX.18 .показана ситуация, когда глобальный контроль приходится на июнь, в этом случае плохо выбирать для локального контроля май, т.к. в нём очень много праздников, в отличие от июня.

Рис. XX.18. Даты обучающей выборки соревнования и итогового теста.

Где используется выбор CV-контроля

Выше мы привели множество способов контроля, оказывается они полезны для самых разных целей в машинном обучении:

- оценка качества модели (об этом была речь выше – мы оцениваем качество по выбранной схеме контроля), этап оценки качества называется **контроль**,
- настройка гиперпараметров (смотрим при каких значениях гиперпараметров качество, оценённое по выбранное схеме контроля, выше), этап настройки гиперпараметров называется **валидация**,
- построение кривых зависимостей качества от параметров: validation curves (от значений гиперпараметров), learning curves (от объёма выборки),
- получение «честных ответов» на обучении (как бы алгоритм отвечал, если бы не обучался на этих объектах), что применяется также при ансамблировании (см. также главу про ансамблирование).
- предобработка данных (подробнее об этом поговорим в соответствующей главе, например для кодирования категориальных признаков).

Кривые обучения (Learning Curves) показывают зависимость качества / ошибки от объёма выборки. Все данные делятся на обучение и контроль (м.б. несколько раз по-разному), обучаемся на $k\%$ от обучающей выборки для разных k и контролируемся на оставшейся части выборки. В результате можем построить график ошибок/качества на train/CV в зависимости от k. Ниже представлено несколько таких кривых обучения: на рис. XX.19 есть зазор между качеством на обучении и тесте, графики продолжают расти в правой части (поэтому добавление данных может увеличить качество). На рис. XX.20 зазора

По кривым обучения можно оценить «достаточность данных».

практически нет, т.е. нет «переобучения» (как мы увидим дальше, это может быть связано с тем, что «модель слишком простая»), все кривые вышли на асимптоту (видно, что достижения предельного качества им достаточно меньше данных – около 50%).

```
train_sizes, train_scores, test_scores =
    model_selection.learning_curve(estimator, X, y,
                                   cv=cv, train_sizes=train_sizes)
```

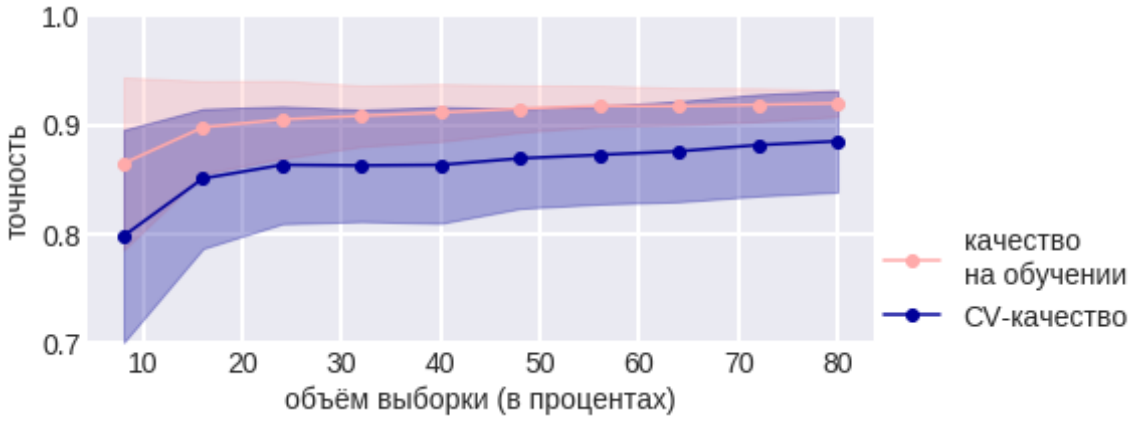


Рис. XX.19. Кривая обучения: переподгонка и недостаток данных.

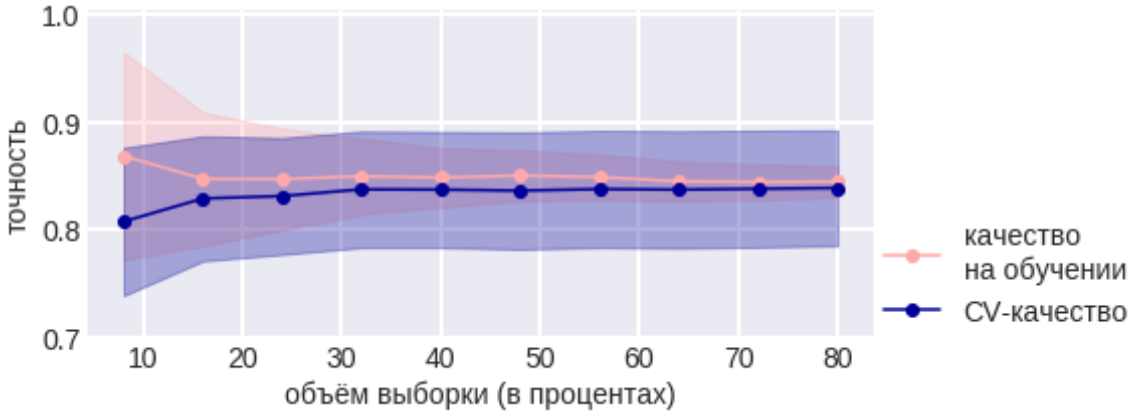


Рис. XX.20. Кривая обучения: недоподгонка и достаточный объём данных.

Валидационная кривая (Validation Curve[1]) показывает зависимость качества / ошибки при выбранной схеме контроля от значений гиперпараметров. По ней хорошо оценивать, как на качество влияет отдельный параметр. На рис. XX.1 показано качество метода kNN при различных значениях гиперпараметра k на обучающей и тестовой выборках.

Перебор значений гиперпараметров также осуществляется с помощью описанных схем контроля (данные делятся на обучение и контроль, м.б. очень много раз). При разных значениях гиперпараметров обучаемся и оцениваем

[1] Функция **sklearn.model_selection.validation_curve** в sklearn.

качество. На рис. XX.21 показан результат перебора числа соседей и метрик в методе kNN. Здесь значения перебирались по сетке: для каждого гиперпараметра задавалось конечное множество значений и были рассмотрены всевозможные комбинации значений гиперпараметров.

```
from sklearn.model_selection import GridSearchCV
parameters = {'metric':('euclidean', 'manhattan', 'chebyshev'),
              'n_neighbors':[1, 3, 5, 7, 9, 11] ,
scoring='roc_auc'}
clf = GridSearchCV(estimator, parameters, cv=5)
clf.fit(X, y)
clf.cv_results_['mean_test_score']
```

	$k=1$	$k=3$	$k=5$	$k=7$	$k=9$	$k=11$
euclidean	76.0	77.0	79.0	78.5	80.5	82.5
manhattan	74.0	74.0	79.0	79.5	80.5	81.0
chebyshev	76.5	78.5	80.0	80.0	81.0	81.5

Рис. XX.21. Перебор значений гиперпараметров по сетке.

Можно осуществлять также случайный поиск[1] (в нём делается ограничение по числу итераций и рассматриваются случайные допустимые комбинации значений гиперпараметров), он считается предпочтительнее для сложных моделей с большим числом параметров, см. рис. XX.22-23 – при переборе по сетке мы исследуем несколько значений каждого параметра, при случайном поиске таких значений больше.

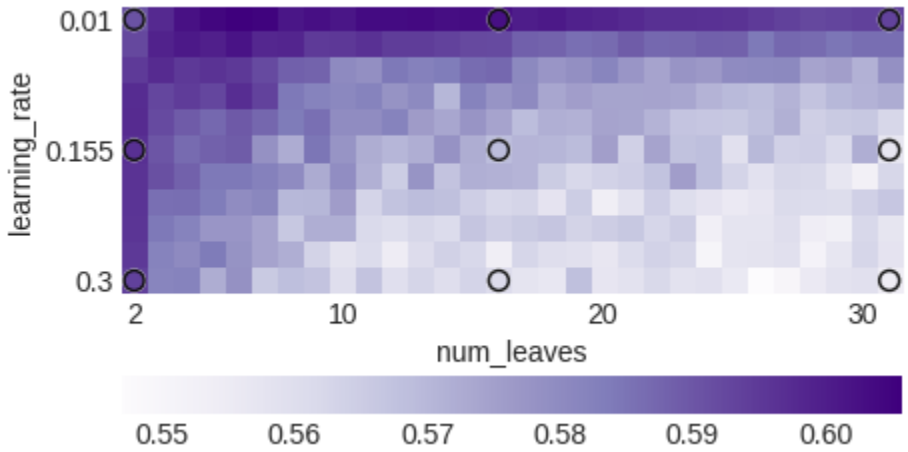


Рис. XX.22. Перебор значений гиперпараметров по сетке GridSearchCV.

[1] В sklearn это функция model_selection.RandomizedSearchCV.

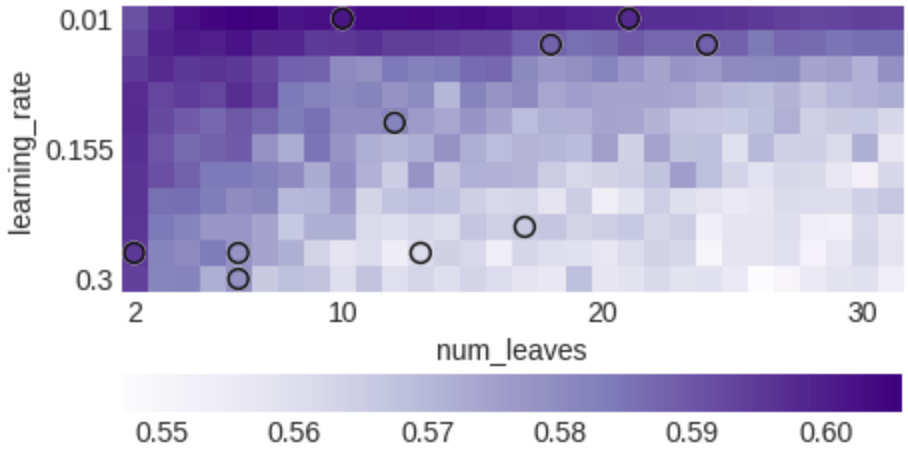


Рис. XX.23. Перебор значений гиперпараметров по сетке RandomizedSearchCV.

Для **получения «честных ответов»** на обучающей выборке выбранной модели алгоритмов обычно используют схему разделения на фолды. Для каждого фолда обучаются на объектах, которые в фолд не попали, а затем получают ответы на данном фолде, см. рис. XX.24. Поскольку каждый объект лежит только в одном фолде, в итоге мы получаем ответы модели на всех объектов выборки. «Честность» ответов заключается в том, что ответ на каждом объекте получен алгоритмом, который не видел метки объекта при обучении.

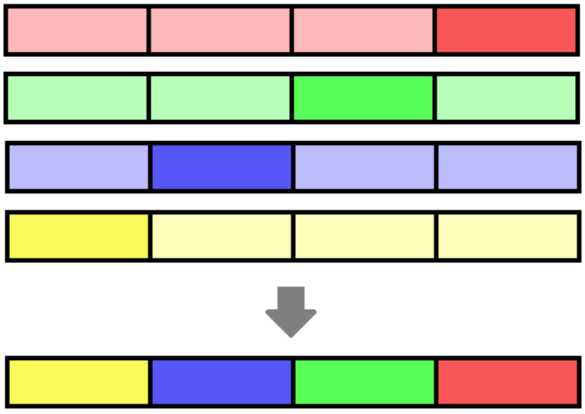


Рис. XX.24. Схема получения «честных ответов».

На рис. XX.25 сравниваются ответы, полученные описанным способом[1], и ответы алгоритма, который был обучен на исходной выборке: алгоритм был обучен на всех данных, а затем дал ответы на всех данных. По обеим осям отложены вероятности принадлежности классу 1 в бинарной задаче классификации, выданные алгоритмами. Видно, что в случае простой модели (слева) честные и нечестные ответы коррелируют, а вот при использовании

[1] Использовалась функция cross_val_predict библиотеки sklearn.

сложного ансамбля (справа) нечестные ответы существенно лучше[1] (для объектов класса 1 выдаваемая вероятность больше 0.5).

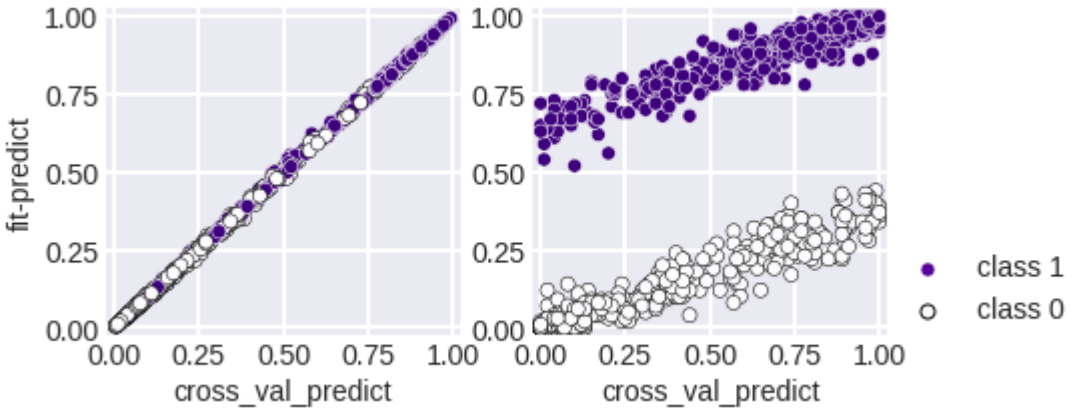


Рис. XX.25. Честные ответы (cross_val_predict) и ответы при обучении на той же выборке (fit-predict) для логистической регрессии (слева) и случайного леса (справа).

Организация контроля: приложения

Приведём примеры правильной организации валидации и окончательного тестирования в реальных проектах. Рассмотрим задачу регрессии, на неё также легко обобщаются многие концепции, описанные в этой главе. Например, **стратифицированный контроль в регрессии**. В задачах классификации при стратифицированном контроле поддерживаются одинаковые пропорции классов в разных подвыборках. В задаче регрессии можно по целевому признаку создать новый служебный признак: номер квантили, в который попало целевое значение, и организовать стратифицированное разбиение выборки для такого признака (считая его целевым), в дальнейшем использовать полученное разбиение для исходного целевого признака. Это гарантирует, что в разные подвыборки в одной пропорции войдут объекты с целевыми значениями из разных диапазонов.

Хороший приём, чтобы не писать лишний код: **подмена задачи**. Делаем вид, что решаем задачу классификации, пользуемся библиотечными функциями, затем возвращаемся к исходной задачи регрессии.

Ещё пример из проекта по прогнозированию продаж: была статистика продаж в разных магазинах крупной сети, необходимо создать алгоритм прогнозирования объёма продаж (на некоторый временной отрезок вперёд).

[1] Можно также вспомнить, что метод ближайшего соседа никогда не ошибается на обучающей выборке.

Прежде чем выбирать модель и заниматься обучением, проанализируем имеющиеся данные. На рис. XX.26 показаны общие продажи компании за несколько лет её существования по дням недели. Каждый график – отдельный год. Видно, что «профиль недели» первого года существенно отличается от профилей остальных лет. Это повод сделать вывод, что первый год очень нетипичный (старт компании) и лучше его совсем исключить из статистики (при общении с заказчиком подтвердилось, что в первый год магазины сети работали в другом режиме). Как мы видим, надо правильно выбирать не только контрольную выборку, **правильная обучающая выборка также важна!**

Для анализа данных очень полезна агрегация по какому-нибудь категориальному признаку.

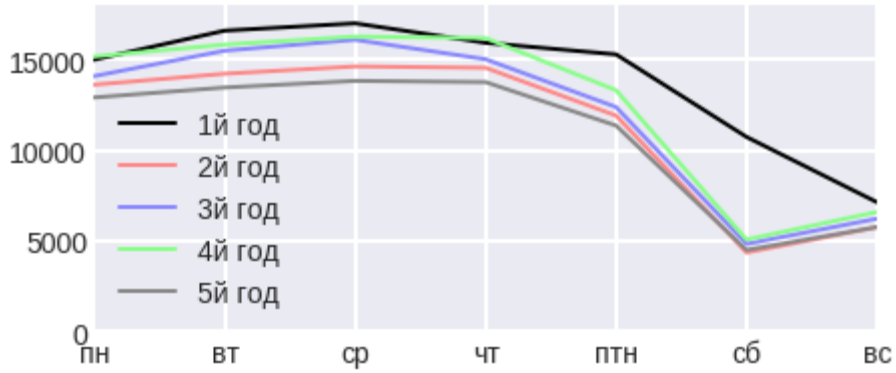


Рис. XX.26. Продажи компании (в штуках) в разные годы по дням неделям.

Ещё интересная история из соревнования хакатона ВТБ[1] про **подгонку локального контроля под глобальный**. В ходе соревнования участники жаловались, что результаты локального контроля не согласуются с глобальным (качеством на отложенной организаторами выборке, которое выводилось на официальном сайте после отправки решений). Это была задача прогнозирования, поэтому:

- локальный контроль должен быть сделан в режиме «out of time»: в контроль попадают объекты после некоторого порога времени t,

также часто бывает, что надо сделать зазор (gap) между обучением и контролем (если такой существует на глобальном контроле): до порога времени t_1 – обучение, после t_2 – контроль, $\text{gap} = t_2 - t_1 > 0$. Затем, можно обратить внимание, что в глобальном контроле доля юридических лиц меньше, чем в данной участникам размеченной выборке, поэтому

[1] https://dsbattle.com/hackathons/mkb/

- следует сделать локальный контроль статистически похожим на глобальный (здесь: удалить часть юридических лиц).

Даже контроль надо обучать.

И наконец, было сделано интересное наблюдение, что изменение временного порога приводит к изменению результатов локального контроля и можно

- подобрать временной порог t так, чтобы результаты локального контроля максимально коррелировали с результатами глобального.

В соревнованиях **важно понять, как организован глобальный тест.** Например, в задаче прогнозирования в глобальный тест вынесены объекты из будущего. При этом при загрузке решения вы видите в турнирной таблице свой результат на какой-то подвыборке глобального теста – публичной части, окончательные результаты объявляются по остальной части – приватной. Организатор не разглашает, как было произведено разделение на публичную и приватную части. Если разделение случайное, то можно предполагать, что публичный результат коррелирует с приватным. Если разделение произошло по времени (обычно «ближайшее будущее» помещают в публичную часть, а «дальнее» – в приватную), то ориентироваться на результаты публичной части для определения качества на приватной нельзя. Можно определить, какое из этих двух разделений применялось. Простейший способ: портить ответы разных временных интервалов и анализировать изменение качества на публичной части.

Один из главных секретов решения любой задачи: знать всё про данные, в том числе про принципы их разделения.

Задачи

1. На рис. XX.17 валидационные кривые показаны вместе с std-коридорами: кривая показывает значение качества при конкретных значениях гиперпараметра, а закрашенная область имеет ширину $2 \cdot \text{std}$, где std – стандартное отклонение, которое оценивается на значениях качества по всем фолдам. Почему при увеличении числа фолдов увеличивается стандартное отклонение? Как оно может зависеть от числа фолдов? Возможна ли зависимость показанная на рис. XX.27?

Рис. XX.27. Зависимость разброса качества от числа фолдов при k-fold CV.

2. Как оценка качества модели и дисперсия оценки может зависеть от объёма обучающей выборки в схеме ShuffleSplit (делаем несколько раз случайное разбиение размеченных данных на обучение и тест, объём обучения задан, в тест попадают все остальные объекты)? Возможна ли зависимость, показанная на рис. XX.28 (пунктиром показано качество на отложенном контроле больших размеров)? Можно ли так на практике определить оптимальный объём обучающей выборки при валидации?

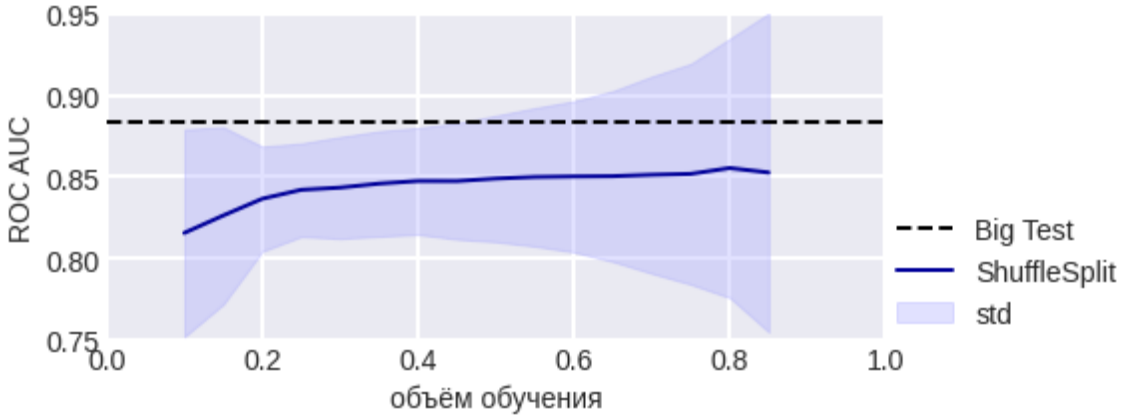


Рис. XX.28. Зависимость оценки качества от доли обучения в ShuffleSplit.

3. Упростим предыдущую задачу. Есть нечестная монета, которая выпадает орлом вверх с вероятностью p и решкой с вероятностью $(1-p)$. У нас есть данные о m бросках, мы разбиваем данные на две непересекающиеся части: train и test, и по каждой оцениваем вероятности выпадения орла: p_{train}, p_{test}. Далее считаем «ошибку» $|p_{\text{test}} - p_{\text{train}}|$. В какой пропорции надо разбивать чтобы 1) ошибка в среднем была наименьшая, 2) разброс ошибки в среднем был наименьший? Почему ответ в этой задаче не согласуется с ответом предыдущей (оптимальные пропорции разбиения получаются другие при оценке качества алгоритмов классификации и регрессии)?

4. Для построения кривой обучения (Learning Curve) берутся обучающие выборки разных объёмов. Как правильнее оценивать ошибку: на фиксированном контроле (также фиксированного объёма) или на всех объектах, которые не попали в обучение?

5. Как организаторы соревнования могут снизить эффективность метода порчи ответов в разных временных интервалах для определения способа разбиения контрольной выборки на публичную и приватную части? Какие есть более сложные способы для решения поставленной задачи?

Итоги

- Организация контроля качества – важный этап решения задачи.
- Базовая идея проверки качества: отложенный контроль.
- Другие варианты организации контроля: перекрёстная проверка, бутстреп, контроль по времени.
- Три правила разбиения выборки: моделирование реальности, недопущение утечки, случайность.
- Стратифицированный контроль сохраняет пропорции классов.
- Объекты могут объединяться в группы, которые не следует разбивать при реализации схем контроля.
- Общее разбиение выборки: обучающая + валидационная + тестовая.
- Схемы разбиения выборки применяются во многих случаях (контроль, валидация, построение кривых, получение честных ответов, предобработка данных).

Код к главе

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
cv = ShuffleSplit(n_splits=3, test_size=0.1,
                  train_size=None, random_state=None)
cross_val_score(logreg, X, y, cv=cv)
```

Код. XX.1. Оценка качества: функция **cross_val_score** проводит оценку качества по нужной схеме на указанных данных.

```
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
cv = KFold(n_splits=10, shuffle=True, random_state=1)
a_rf = cross_val_predict(rf, X, y, cv=cv) # ответы rf на CV
a_gbm = cross_val_predict(gbm, X, y, cv=cv) # ответы gbm на CV

plt.scatter(a_rf, a_gbm, c=y)
plt.xlabel('rf')
plt.ylabel('gbm')
```

Код. XX.2. Получение ответов алгоритма: проводится указанное разбиение данных на фолды, запоминаются ответы на каждом фолде и конкатенируются в один вектор – это выход функции **cross_val_predict**.

Спасибо за внимание к книге!
Замечания по содержанию, замеченные ошибки и неточности можно написать в телеграм-чате
https://t.me/Dyakonovsbook